1 Job Scheduling

1. Sort job queue
2. Reserve a order
3. Try to backfill
4. when reserved order ends,loop from 1 (the first one in the sorted queue starts immediately, no reservation)

conditions for backfilling:

1. It will terminate by the shadow time, OR
   - 'Shadow time' refers to the expected start time of the first queued job
2. It needs at most extra PEs
   - 'Extra PEs' refers to the number of available PEs when the first queued job starts running

1.1 Backfilling variations

- Order of queued jobs: FCFS (EASY backfilling), priorities (slack-based backfilling), shortest estimated runtime first (SJF backfilling)
- Number of reserved jobs: 1 (EASY backfilling), all (conservative backfilling), adaptive (selective backfilling: only reserve jobs whose expected slowdown exceeds a threshold)

2 Gang Scheduling

1. assign order by order (following the policy and the queue order), until a quantum of each job has been assigned
2. if there is no repacking, repeat same pattern until all jobs end
3. if there is repacking, every time a job terminates re-schedule following queue (but maybe maintaining affinities)

3 Metrics for Job Scheduling

- Wait Time (T_W)
- Response Time ($T_W + T_R$)
- Slowdown ($(T_W + T_R)/T_R$)
- Bounded Slowdown ($max\{(T_W + T_R)/max\{T_R, \tau\}, 1\}$)
- Per-Processor Slowdown ($slowdown$/number of PE)

4 Memory Virtualization

- Page Table: from host Virtual memory to host Physical memory.
- Shadow Page: from host Virtual memory to guest Physical memory.

5 Borrowed Virtual Time

- For each $vCPU_i$, track its virtual time (A_i): $A_i + = \text{RealRunningTime} \times m_i$
  - Real running time measured in multiples of the minimum charging unit (mcu), typically the timer interrupt period.
  - m_i is set proportional to $\frac{1}{w_i}$ (with w_i being the weight of $vCPU_i$), and normalized to be an integer.
- Switch from i to j if $A_j \leq A_i - C \times m_i$:
  - C is the context switch allowance. Real time by which the current $vCPU$ is allowed to go beyond another runnable $vCPU$ (it is a multiple of the mcu)
  - Check only $A_j \leq A_i$ (C is ignored) if j just became runnable after sleep to avoid affecting response time.

6 Pre-Copy VM migration

1. Pre-migration + reservation
   - Determine migrating VM
   - Determine destination host
   - Initialize a VM on the target host
2. Memory push (iterative pre-copy)
   - Transfer all memory pages
   - Copy dirty memory pages in successive rounds
   - Repeat until the number of dirty pages is smaller than a threshold
3. Stop and copy
   - Suspend VM on the source host
   - Network connection is redirected to the new VM
   - Transfer the remaining dirty pages
   - Transfer VCPU and network states
4. Commitment + activation
   - VM reloads state and resumes its execution on the destination host
   - Remove original VM from source

7 Post-Copy VM migration

1. Pre-migration+reservation is the same
2. Stop and copy
   - Suspend VM and transfer minimal execution state
3. Activation: resume VM at destination
4. Running VM
   - During the execution of the VM, its memory pages are pushed from the source host
     - If VM accesses a not yet received page, it is faulted in from the source over the network (remote page fault)
   - Remove the original VM from the source host when all the pages have been transferred

8 Primary Backup protocols

8.1 remote-write

- All writes are done at a fixed single replica
- Reads can be carried out locally
- sequential consistency: Primary send all writes to each replica in the same order via total+FIFO-order atomic multicast
- Implements strong consistency if read requests are also forwarded to the primary

8.2 local-write

Primary migrates to the replica that is writing, successive writes are carried out locally, and then the replicas are updated using a non-blocking protocol

9 Replicated-write protocols

9.1 Active Replication

- Each operation is forwarded to all replicas
  - Each replica has a process to carry out operations
- Operations need to be carried out in the same order everywhere
  - Client uses total+FIFO-ordered multicast to send operation to the group of replicas
  - Total+FIFO-ordered multicast ensures:
    * Sequential consistency if we multicast only writes, and
    * Strong consistency if we multicast also reads

10 Quorum-based Protocols

Writes are serialized and reads return the latest version that was written $\implies$ Strong Consistency

11 Block vs Object File System

Block storage device:

- Data stored in fixed-size blocks
- Identified by a positive integer starting from 0
- LBN used for data retrieval
- Block-based file system → store files

Object storage device:

- Data is a single object
- Key & Value
- Key → Positive integer (64 bits)
- Value → Arbitrary size
- Attributes
- Object-based file system → store files

12 Memcached

- In-memory key-value store for small arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.
- It is a client-server model
  - Client knows how to access the servers
  - Servers store the values
  - No communication between servers (Client-server only)
  - LRU cache (Values expire after a specified amount of time)
- Consistent Hashing for keys (Model that allows for more stable distribution of keys given addition or removal of servers)
- Under the hood, it uses a slab allocator
- Memcached memory → 1MiB/page
- Pages associated with slab classes
- Slab class → Fixed-sized chunks
- #chunks/slab class == #pages/sizeof(slab class)

13 Redis

- primary based remote-write protocol.
- Master/replicas replication is Asynchronous, Non-blocking
- Allow writes only with N attached replicas
- Redis expires allow keys to have a limited time to live.

All write operations are centralized in a primary replica (always the same). Primary replica is in charge of coordinating write operations to the data item. In Redis, write operations on a different replica than the primary are disabled (therefore all of them are local and the forwarding to the primary is never needed). Write Requests from clients at primary replicas are acknowledged and propagated in the background to remaining replicas.

13.1 commands

- Start master using default port (6379): masterIP$ src/redis-server
- Start 1 replicas at node1IP$: node1IP$ src/redis-server –replica-of masterIP 6379
- min-replicas-to-write
- ./src/redis-cli -p port$ get/set/del key
- sentinel monitor name$ ip$ port$ quorum$

14 CouchDB

- Document-based database
- Fault-tolerant
- Easy scaling up/down
- Incremental replication
- Eventual consistency

Cluster configuration:

- Q: Number of shards
  - Shard ∼ part of database (→ 4 shards == 4 nodes)
- N: Number of replicas (copies of every document)
- R: Number of copies of a document with the same revision to be read before returning it
- W: Number of nodes needed to save the document

15 MongoDB

- MongoDB is a document database that provides high performance, high availability, and automatic scaling.
- Document-based database
- Fault-tolerant
- Easy scaling up/down
- Incremental replication
- Eventual consistency

16 IBM General Parallel File System

- High-performance clustered file system
- Posix Compliant
- Directory + Metadata distributed across the file system
- File split in blocks of 1MiB
- No replication
- Tolerates network partitions (The largest partition remains live)
- Used in HPC

17 Dropbox

Dropbox keeps a desktop folder synchronized with a server in the cloud. The main components are:

- Block Server - To store blocks from files in Amazon S3
- MetaData Server - To store info about files in a DB
- Notification Server - To notify clients about new changes and avoid polling.

Due to the high number of requests, all these components must be replicated (notification server, metadata server, block server) and different load balancers are used to distribute the load against the available servers. Additionally, to avoid the access to the parallel DB for the metadata it is kept in memory through an in-memory database: 'memcached'. The clients access the metadata servers through the load balancers, who consult the in-memory metadata of memcached, and allows direct access to the block server with the data in amazon. Any change in the metadata/block is notified to the notification server who is responsible to forward the notification to the different clients.

18 Google File System

A distributed file system developed by Google to provide efficient, reliable access to data using large clusters of commodity hardware.

- Fault tolerance: "Failure is the norm, not the exception"
- Files are huge
- Most files appended, not overwritten
- Read over write ratio (Very High!)

- GFS does not have a standard OS-layer API
  - NO Posix API
  - No kernel/vfs for accessing files
  - User-level API to access files
  - GFS servers are implemented in user space using native Linux FS
- Files organized in directories
- Operations:
  - Basic: Create, delete, open, close, read, write
  - Additional: Snapshot & append

18.1 Chunkservers

- Chunk size = 64 MB (default)
  - Chunkserver stores a 32-bit checksum with each chunk
    - ⁎ In memory & logged to disk: allows it to detect data corruption
- Chunk Handle
  - Globally unique 64-bit number
  - Assigned by the master when the chunk is created
- Each chunk is replicated on multiple chunkservers
  - Three replicas (different levels can be specified)
  - Popular files may need more replicas to avoid hotspots

18.2 One master

- Maintains all file system metadata in memory
  - Namespace
  - Access control info
  - Filename to chunks mappings
  - Current locations of chunks
- Manages
  - Chunk leases (locks)
  - Garbage collection (freeing unused chunks)
  - Chunk migration (copying/moving chunks)
- Master replicates its data for fault tolerance
- Periodically communicates with all chunkservers
  - Via heartbeat messages
  - To get state and send commands

18.3 Client

- Clients connect directly with chunkservers
- No data caching at server/client
- Clients cache metadata

18.3.1 Reading files

1. Contact the master
2. Get the list of chunk handles (and files metadata)
3. Get the location of each of the chunk handles (replicated info)
4. Contact ANY available chunkserver

18.3.2 Writing to file

Control phase (send but don't write):

1. A client is given a list of replicas
2. Client writes to closest replica
   - Replica forwards data to another replica
   - That replica forwards to another chunkserver
3. Chunkservers store this data in a cache

Data phase (write):

1. Client waits for replicas to acknowledge receiving data.
2. Send a write request to the primary
3. The primary is responsible for serialization of writes (assigns consecutive serial numbers to all writes)
4. Once all acks have been received, the primary acknowledges the client

19 Hadoop File System: Storage system

- Run on commodity hardware
- Fault-tolerant
- High throughput vs low latency
- No Posix Compliant
- Support large data sets
  - File size of GB – TB & 100s servers
- Write-once-read-many access model for files
- A file´s content need not be changed
  - Except for appends and truncates
- Suitable for MapReduce apps or web crawlers
- Simplifies data coherency & enables high throughput
- NameNode server
  - Manages the file system namespace
  - Stores informations about files (names, attributes)
  - Regulates access to files by clients
- N DataNodes

19.1 NameNode

- Manages FS namespace operations:
  - Open, close
  - Renaming files & directories
  - Mapping of blocks to DataNodes
- Heartbeat and Blockreport from DataNodes
  - Detect malfunctions
  - List of all blocks on a DataNode

19.2 DataNodes

Manage storage attached to the node

- Serving Read & Write requests from File Syatem's clients
- Block creation, deletion & replication (when requested from NameNode)
- Store blocks' files in the local file system (configurable block size, default 128Mb)
- Blocks are replicated, #replicas per file is configurable

19.3 Replicas

Replica placement (Rack-Aware):

- 2 in different nodes in local rack
- 1 in different rack
- remaining in random places keeping replicas/rack below a threshold

Replica selection:

- Client sends request to NameNode, receives list of blocks and replica DataNodes per block
- Client tries to read from the closest replica

19.4 commands

- start server: sbin/start-dfs.sh
- check up services: jps
- bin/hdfs dfs -copyFromLocal <local file path> <dest(present on hdfs)>
- set n replicas: dfs.replication=n
- modify block size: dfs.blocksize
- `hadoop jar jar_file main_class input_path output_path`: Runs a Hadoop MapReduce job.
- `mapred job -list`: Lists MapReduce jobs.

20 Cassandra: a distributed key-value store

- Cassandra follows a Replicated-write quorum-based protocol.
- A peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster
- Main elements:
  - Node: where you store your data
  - Datacenter: collection of nodes
    - ⁎ Granularity used by replication
    - ⁎ Different workload → use separate datacenters
    - ⁎ Datacenters must never span physical locations
  - Cluster: 1 or more datacenters
  - Cassandra is aware of network topology
- Node: where you store your data
- Datacenter: collection of nodes
  - Granularity used by replication
  - Different workload → use separate datacenters
  - Datacenters must never span physical locations
- Cluster: 1 or more datacenters

20.1 Consistent Hashing

- Distribute data across a cluster to minimize reorganization when nodes are added or removed.
- Use a consistent keyspace → a ring
- Data has a partition key
- Each partition key → a hash value into ring
- Each node in the cluster → range of data based on hash value
- Adding or removing a node → Redistribute the data in the interval
- Cassandra places data on each node according to the partition key and the range that the node is responsible for.

20.2 How does Cassandra work

- Client read or write requests can be sent to any node in the cluster.
- When a client connects to a node with a request, that node serves as the coordinator for that particular client operation.
- Coordinator acts as a proxy between client application and nodes that own the data being requested.
- Coordinator determines which nodes in the ring should get the request based on how the cluster is configured.

20.3 Writing Data

- Coordinator sends write request to all replicas.
- Write consistency level determines how many replica nodes must respond with a success acknowledgment for the write to be considered successful.
- Success means that the data was written to the commit log and the memtable.
- A sequentially written commit log on each node captures write activity to ensure data durability.
- Data is then indexed and written to an in-memory structure, called a memtable.
- Each time the memory structure is full, the data is written to disk in an SSTable data file.
- All writes are automatically partitioned and replicated throughout the cluster.

20.4 Data Replication

- Replication factor: total number of replicas across the cluster
  - 1 → only one copy of each row in the cluster
  - 2 → 2 copies of each row, each copy on a different node.
- Replica placement
  - Partitioner: determines which node will receive the first replica of a piece of data, and how to distribute other replicas across other nodes in the cluster (The hash function)
  - Replica placement strategy: determines which nodes to place replicas on.
    - ⁎ SimpleStrategy: Single datacenter and 1 rack.
    - ⁎ NetworkTopologyStrategy: Remaining cases.

20.5 Consistency models

Fully configurable consistency on: cluster, datacenter or individual read or write operation

- By default: ONE for all read & write ops
- QUORUM = $\left(\frac{\text{SumReplicationFactors}}{2}\right) + 1$

20.5.1 write consistency models

- ALL: ...all replica nodes in the cluster for that partition
- QUORUM: ...a quorum of replica nodes in...
  - EACH_QUORUM: EACH datacenter.
  - QUORUM: ALL datacenter.
  - LOCAL_QUORUM: SAME datacenter as coordinator
- ONE: ... at least one replica node
- TWO: ...at least two replica nodes
- LOCAL_ONE: ... at least one replica node in the same datacenter
- ANY: ... at least one node (note the missing "replica"part!)

20.5.2 read consistency model

same as writing, except for EACH_QUORUM and ANY

20.6 commands

- `nodetool status`: Displays node status and cluster info.
- `nodetool ring`: Visualizes data distribution across nodes.
- `cqlsh`: Starts CQL shell for commands.
- `DESCRIBE KEYSPACES;` Lists all keyspaces.
- `USE keyspace_name;` Switches to a keyspace.
- `DESCRIBE TABLE table_name;` Shows table structure.
- `INSERT INTO table_name ...;` Inserts data.

21 metrics for Energy Management

- PUE: $TotalPower/IT\ Power$
- SPUE: $ITPower/Server\ Input\ Power$
- TPUE: $PUExSPUE$
- DCeP: $TotalEnergy/Useful\ Work$
- $P_{idle} * \delta_{idle} > E_{OFF} + E_{ON} + P_{OFF} * (\delta_{idle} + \delta_{OFF} + \delta_{ON})$